A Multiple-Domain Evaluation of Stratified Case-Based Reasoning

L. Karl Branting and Yi Tao
Department of Computer Science
University of Wyoming
Laramie, WY, USA
karl@uwyo.edu

Abstract. Stratified case-based reasoning (SCBR) is a technique in which case abstractions are used to assist case retrieval, matching, and adaptation. Previous work has shown that SCBR can significantly decrease the computational expense required for retrieval, matching, and adaptation under a variety of different problem conditions. This paper extends this work to two new domains: a problem in combinatorial optimization, sorting by prefix reversal; and logistics planning. An empirical evaluation in the prefix-reversal problem showed that SCBR reduced search cost, but severely degraded solution quality. By contrast, in logistics planning, use of SCBR as an indexing mechanism led to faster solution times and permitted more problems to be solved than either hierarchical problem solving (by ALPINE) or ground level CBR (by SPA) alone. The primary factor responsible for the difference in SCBR's performance in these two domains appeared to be that the *optimal-case utility* was low in the prefix-reversal task but high in logistics planning.

1 Introduction

Human problem solvers exhibit great flexibility in reasoning both with specific cases and with abstractions derived from one or more cases [13]. A number of case-based reasoning systems have modeled an important aspect of this flexibility: use of case abstractions to guide case indexing, matching, and adaptation. Termed *stratified case-based reasoning* (SCBR) in [4], this approach has been used for planning [3], [12], design of control software [16], and route planning [4], [5]. A comparative analysis of various approaches to using abstraction in CBR is set forth in [2].

Systematic empirical analyses set forth in [4] and [5] compared the performance of heuristic search (A*), *Refinement* [11] (*i.e.*, hierarchical problem solving), ground-level CBR, and SCBR as a function of (1) number of levels of abstraction, (2) the size of the case library, (3) resemblance among cases, and (4) whether the abstraction hierarchy satisfies the downward-refinement property [1]. However, this evaluation was limited to a route-finding domain in which there was a high liklihood, for typical problems and case libraries, that the case library would contain a case that could be adapted to a solution to the problem as good as would be obtained through *ab initio* problem solving. This liklihood, which depends on both the adaptation method and the case library coverage, is termed the *optimal-case utility* of the library and adaption method.

This paper describes two sets of experiments in which SCBR was applied to alternative domains. Section 2 describes the application of SCBR to a problem in combinatorial optimization, sorting by prefix reversal. Section 3 describes a prototype application of SCBR to logistics planning. These experiments suggest that optimal-case utility is a key factor in the applicability of SCBR.

2 SCBR for Combinatorial Optimization

2.1 The Prefix-Reversal Problem

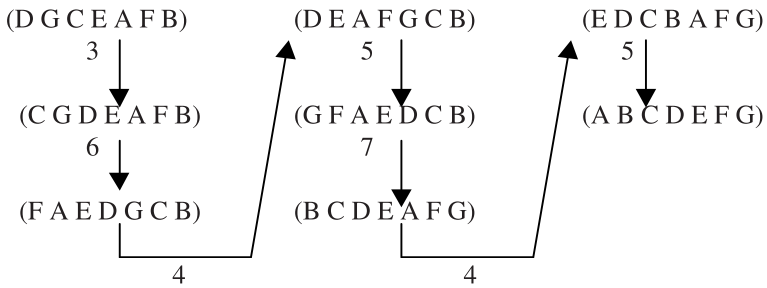
S*orting by reversals* is the task of determining the minimum number of subsequence reversals necessary to transform one sequence into another. This task, which arises in molecular biology [15], has been shown to be NP-hard in the general case [6]. *Sorting by prefix reversals* is a special case of this problem in which all reversals must be of sequence prefixes. Bounds for sorting by prefix reversals were derived in [8].

A prefix-reversal problem is given by specifying start and goal states, each represented as a list of elements. An n-prefix-reversal problem is a prefix-reversal problem involving permutations of an n-element sequence. For example, a 7-prefix-reversal problem is as follows:

Start: (D G C E A F B)

Goal: (A B C D E F G)

A solution is a sequence of operations, where each operator represents the length of the prefix being reversed. For example, the solution to the problem above is (3 6 4 5 7 4 5):



2.2 Abstraction for Prefix Reversal

A simple abstraction mechanism for the prefix-reversal problem is to simply drop from the representation the object that occurs last in the goal state, thereby aggregating all states that differ only in the position of this object into a single state. A solution to the original problem can't be any shorter than the solution to an abstraction of that problem, so the solution length of the abstraction is an admissible distance heuristic, h*, for A*. Moreover, if a solution is obtained for the abstract problem, then at most 4 additional operations are required to solve the original problem. This is because if there are n objects, the operations (p, n, n-1, p-1) will move the pth object to last place without altering the position of any other objects.

Refinement search [11] can be performed using this abstraction hierarchy by solving an abstraction of the original problem, then using this solution as a heuristic for search at the next lower level of abstraction. This process is repeated until the ground level is reached. A very strong alternative heuristic for this problem counts the number for pairs adjacent in the goal state that are not adjacent in the current state. This *adjacent-pairs* heuristic is also admissible, because a prefix reversal can reduce the number of such pairs by at most one.

2.3 Algorithms

To evaluate the effectiveness of SCBR at reducing search in this domain, four different approaches were compared: A*, using the adjacent-pairs heuristic; refinement; ground-level CBR (GRCL), which adapts every case in the library and returns the one with the shortest adapted solution; and Reuse-Closest (RCL), an SCBR algorithm that starts with the most specific matching cases (or the most abstract cases, if no cases match), finds the refinements of each case, adapts each refinement (using A* to find the shortest adaptation paths from the start and goal positions to the solution path at that level of abstraction), and selects the refinements having the shortest adapted solution paths. All four algorithms are described in detail in [4].

As in the route-finding problem described in [4], adaptation in RCL consisted of using A* to find the shortest paths from the start and goal positions to the solution path at the current level of abstraction to form a solution consisting of these paths concatenated with a reused segment of the old solution. Thus, an adapted solution must include a portion of the original solution.

2.4 Experimental Evaluation

The dependent variables of interest were search cost, as measured by the number of nodes expanded by A*, and the solution quality, as measured by solution length. The independent variables were:

- The number of abstraction levels (1,2,3,4)
- The size of case library (1,5,10,50,100,500,1000)

Experiments were performed with 8-prefix-reversal, and results were based on 1000 random test cases.

Varying the number of abstraction levels In the first experiment, the number of cases in case library was fixed at 100 and the number of abstraction levels was varied from 1 to 4. As shown in Figure 1, the number of nodes expanded by A* and GRCL remained constant since these algorithms do not make use of the abstraction hierarchy. Refinement performed slightly better than A*, indicating that abstract solution length is a good heuristic in this domain. RCL expands the smallest number of nodes, given 3 or more abstraction levels.

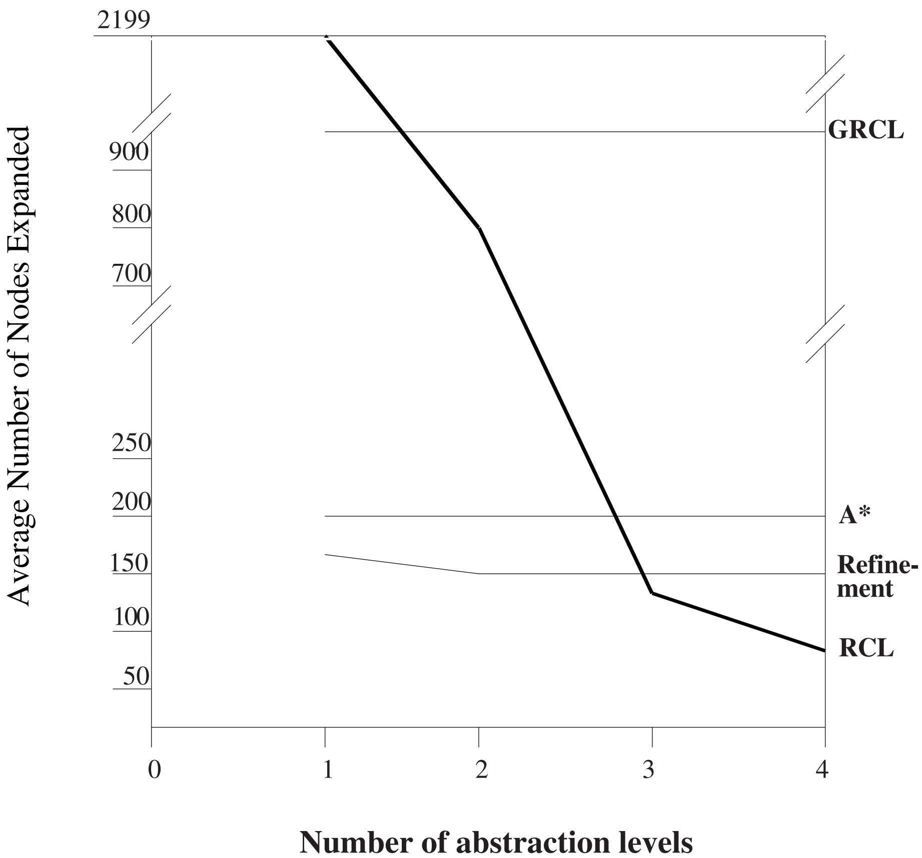


Fig. 1. Mean nodes expanded in 8-prefix-refersal as a function of the number of abstraction levels.

Varying the Case Library Size Figure 2 shows the effect of varying the size of the case library. The number of nodes expanded by A* and Refinement remained constant, because these algorithms do not use cases. The number of nodes expanded climbed steadily for GRCL, but was relatively constant for RCL. Unfortunately, the smallest number of nodes was expanded by GRCL with a case library consisting of a single case. Evidently, a case library consisting of a single case reduces search by converting the original problem of searching for a path to a single state into two easier problems, each consisting of a search for a solution path consisting of, on average, about 7.7 states.

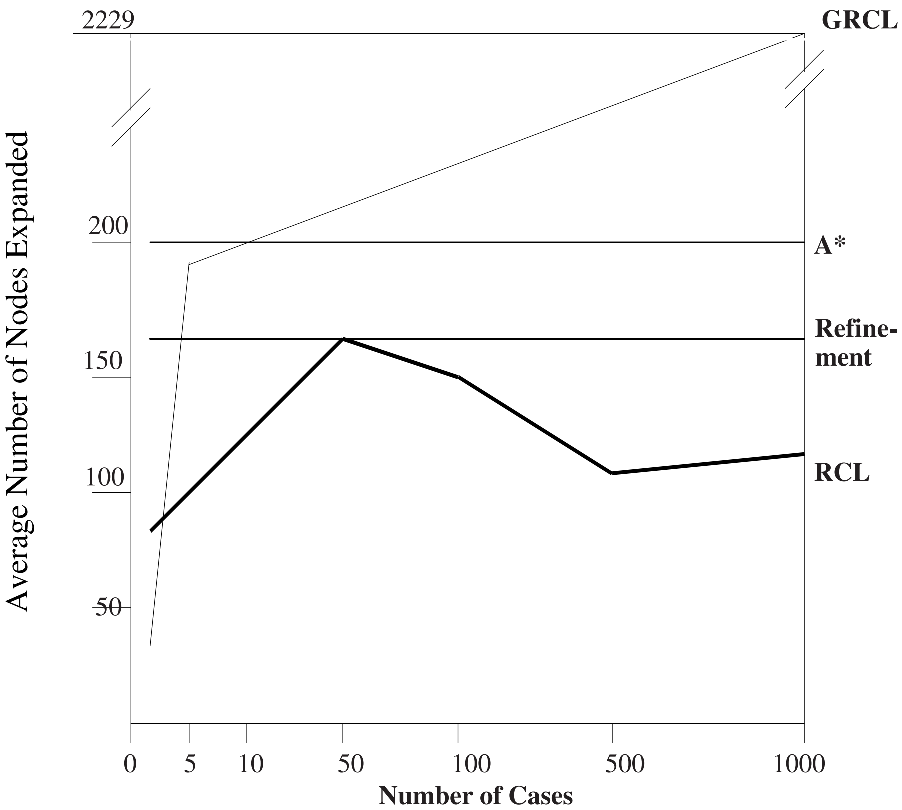


Fig. 2. Mean nodes expanded in 8-prefix-refersal as a function of the number of cases.

Solution Length Tables 1 and 2 show mean solution lengths as a function of abstraction levels (100-case library) and case library size (3 levels of abstraction), respectively.

Table 1. Mean Solution Lengths for 8-prefix-reversal as a function of the number of abstract levels

	A*	HSOLVE	GRCL	RCL
1 abstract level	7.7	7.7	9.3	9.5
2 abstract levels	7.7	7.7	9.3	10.1
3 abstract levels	7.7	7.7	9.3	9.9
4 abstract levels	7.7	7.7	9.3	10.0

Since the adjacent-pairs heuristic is admissible, A* always finds a shortest solution path. Moreover, Refinement also always finds the shortest path because it uses A* at every level, and the length of the next more abstract solution is itself an admissible heuristic. GRCL is guaranteed to return the case requiring the least adaptation. However, the consistent value of 9.3 for the mean solution length obtained by GRCL indicates that the adaptation available to GRCL and RCL (splicing paths onto a reused portion of the old solution) leads to decreased solution quality, even when applied to the most adaptable case in the case library. This is because both problems and solutions are uniformly distributed through state-space in the prefix-reversal problem, so the likelihood that an optimal solution to a new problem will involve reuse of a segment of an old solution is quite low, even if the number of cases is high.

Table 2. Mean Solution lengths for 8-prefix-reversal as a function of the number of cases

	A*	**HSOLVE**	**GRCL**	**RCL**
1 case	7.7	7.7	10.5	10.5
5 cases	7.7	7.7	10.0	10.0
10 cases	7.7	7.7	10.1	10.1
50 cases	7.7	7.7	9.5	10.0
100 cases	7.7	7.7	9.3	9.9
500 cases	7.7	7.7	8.9	10.0
1000 cases	7.7	7.7	8.7	9.6

Since RCL can produce solutions no better than those produced by GRCL, the weakness of the adaptation method guarantees that RCL cases are suboptimal as well. Moreover, while RCL's use of abstract cases to guide retrieval was much less expensive than GRCL's exhaustive matching, the higher mean solution length for RCL means that it did not always find the case leading to the shortest solution.

2.5 Summary

The prefix-reversal experiment illustrated that SCBR can reduce search in a combinatorial optimization problem. However, the experiment also illustrated that SCBR cannot compensate for a weak adaptation method that is incapable of producing solutions as good as those found through *ab initio* problem solving.

3 Logistics Planning

The utility of using case abstractions for indexing and adaptation has been demonstrated in planning domains characterized by task-decomposition abstraction hierarchies [3], [12]. However, development of task-decomposition hierarchies is, in general, a difficult and time-consuming task. The second experiment was designed to explore the feasibility of using SCBR as an indexing technique using inexpensive 'off-the-shelf" abstraction hierarchies. In the early 1990s, several Ph.D. projects developed techniques for automated creation of abstraction hierarchies in the STRIPS formalism, *e.g.*, [8], [14]. One such system, ALPINE, is available to researchers as part of the PRODIGY [7] release version 4 (www.cs.cmu.edu/afs/cs/project/ai-repository/). Alpine is guaranteed to produce abstraction hierarchies with the *ordered monotonicity property* [14], a weaker condition than the downward refinement property shown in [5] to contribute to (although not to be essential to) the effectiveness of SCBR.

An evaluation of the relative contribution of hierarchical problem solving and CBR to SCBR would have the most information value if it involved an ablation of each component, that is, if the hierarchical problem solving, ground-level CBR, and ground-level *ab initio* problem solving components were tested both in isolation and in combination. However, writing a planning system that embodied every combination of these factors was a task beyond the scope of this exploratory project. We therefore selected SPA ([10], a least-commitment generative planner with very simple and general case-retrieval and adaptation mechanisms, as our main planning engine. SPA has no mechanism for hierarchical problem solving, however, so we used ALPINE as our hierarchical problem solver.

The logistics problem domain, taken from the PRODIGY version 4.0 release, includes 14 predicates and 6 operators in STRIPS notation. Problems in this domain involve transportation of objects between various locations through a combination of truck and airplane operations. A logistics transportation problem is given by initial state and a goal state specification. The description of an initial (or any other) state is composed of a list of objects and their corresponding types together with a set of instantiated predicates (i.e. literals) that describes the configuration of those objects. For example, Problem 5 in our test set can be depicted graphically as follows:

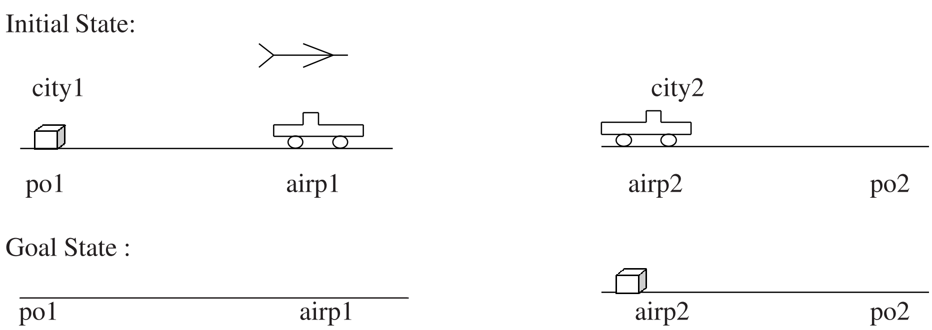


Fig. 3. A Logistics Transportation problem

In STRIPS notation, the problem is as follows:

```
(init
          `((object o1)
          (pos-office po1)
          (pos-office po2)
          (airplane p1)
          (truck t1)
          (truck t2)
          (airport airp1)
          (airport airp2)
          (city c1)
          (city c2)
          (loc-at airp1 c1)
          (loc-at po1 c1)
          (part-of t1 c1)
          (loc-at airp2 c2)
          (loc-at po2 c2)
          (part-of t2 c2)
          (same-city airp1 po1)
          (same-city po1 airp1)
          (same-city airp2 po2)
          (same-city po2 airp2)
          (at-obj o1 po1)
          (at-truck t1 airp1)
          (at-truck t2 airp2)
          (at-airplane p1 airp1)))
(goal `( (at-obj o1 airp2)))
```

One plan for this problem is :

```
          (DRIVE-TRUCK T1 AIRP1 PO1)
          (LOAD-TRUCK O1 T1 PO1)
          (DRIVE-TRUCK T1 PO1 AIRP1)
          (UNLOAD-TRUCK O1 T1 AIRP1)
          (LOAD-AIRPLANE O1 P1 AIRP1)
          (FLY-AIRPLANE P1 AIRP1 AIRP2)
          (UNLOAD-AIRPLANE O1 P1 AIRP2)
```

3.2 Abstraction Hierarchy Creation

ALPINE's problem-independent abstraction hierarchy creating algorithm, described in [14], was applied to various sets of logistics problems. Unfortunately, most resulted in hierarchies with only a single abstraction level. This illustrates a general pitfall of techniques, like SCBR, that used abstraction hierarchies: techniques for automated creation of abstraction hierarchies are still in their infancy. Eventually, however, a set of 9 training cases and 14 testing cases were selected for which ALPINE created a 4-level abstraction hierarchy:

```
(Abstraction
  (Static = loc-at same-city)
  (Level-2 = at-obj
             inside-truck
             inside-airplane)
  (Level-1 = at-airplane)
  (Level-0 = at-truck)
  :order
  (Level-2 > Level-1 Level-0))
```

3.3 Case-Library Creation

Ground level cases for the case library were created using the SPA function *plan-from-scratch.*.Abstract cases were then created bottom-up by the following algorithm:

```
Function AbstractCases (initial, goal, levels, hierarchies)
  cases := nil;
  plan := plan-from-scratch (initial, goal);
  case := make-case(initial, goal, plan);
  push(case, cases);
  for i:=1 to levels-1 do
     initial := drop-literals(initial,first(hierarchies));
     goal    := drop-literals(goal,first(hierarchies));
     plan    := fit-plan (initial, goal plan);
     case    := make-case(initial, goal, plan);
     push(case, cases);
     pop(hierarchies);
  Return cases;
```

As in the route-finding domain, logistics cases with initial and goals that are distinct at one level of abstraction may have identical initial and goal states at higher levels of abstraction. The case library may therefore be organized as a forest as described in [4] and illustrated in Figure 4. Note that abstraction is over states, and that abstract plans are formed by adapting lower-level plans to solve the abstract problem.

3.4 Case Retrieval

In SCBR, the retrieval process starts at the most abstraction level of the case library. After the best-matching of the most abstract cases is determined, the matching process is repeated with the children of the best case until the ground level is reached. In this

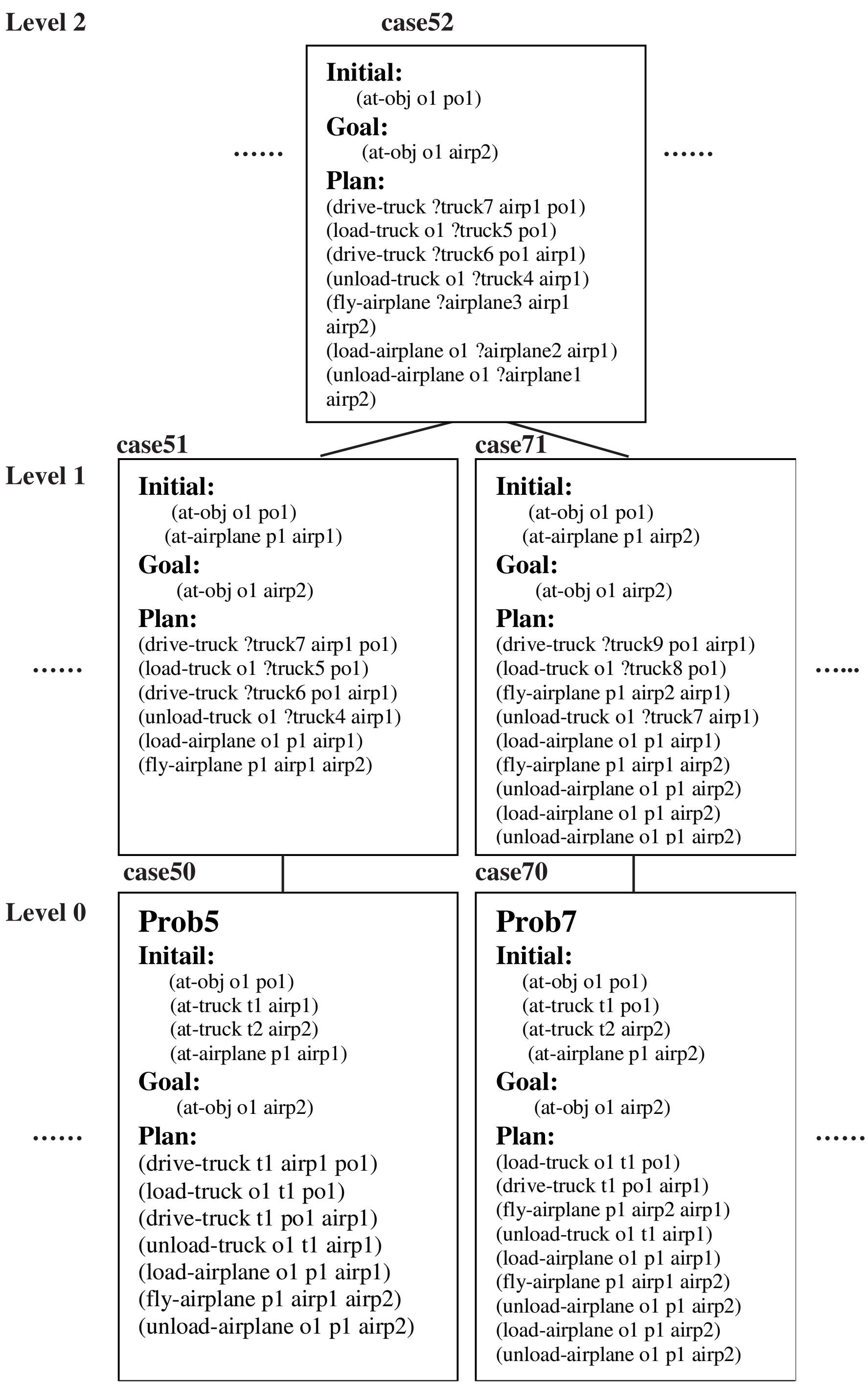


Fig. 4. Case Library Structure

application of SCBR to planning, the retrieval procedure at a given abstraction level consists of two steps. First, the goals of new problem are matched against the goals of each of the set of cases, and the case or cases with the greatest number of matches are identified. If there are several cases whose goals match equally well, the SPA procedure *fit-plan* is applied to each, and the candidate with the fewest open conditions is chosen as the best. The process is repeated for the children of the current best match until the ground level is reached. See [18] for details.

3.5 Experimental Procedure

In this experiment, 6 different planning algorithms were compared.

- **ALPINE**. ALPINE performs hierarchical problem solving.
- **SPA.** Ground-level, *ab initio* planning using the SPA procedure *plan-from-scratch.*
- **SPA-cbr.** Ground-level cbr using *plan-from-library,* SPA's case-based planner.
- **SPA-cbr-cascading**. SPA-cbr with learning, i.e., each new case is added to the case library.
- **SCBR**. Uses the procedure described above for indexing and *fit-plan* for adaptation
- **SCBR-cascading**. SCBR with learning.

Solvability It quickly became apparent that many of logistics problems were not solvable by all the algorithms within a reasonable time (less than 2 hours) even on a large lisp server (indeed, the greatest barrier to an empirical evaluation of SCBR in this domain was simply accumulating a sufficiently large corpus of problems that could be solved by an *ab initio*, ground-level planner). The first experiment tested the ability of each algorithm to solve the 14 test problems.

The 9 training cases, selected because they could all be solved by SPA *plan-from-scratch* procedure, were given as training data to the CBR planners, SPA-cbr, SPA-cbr-cascading, SCBR, and SCBR-cascading. Fourteen test cases (set forth in [18]) were then presented, in order, to each algorithm. Problems that were not be completed in 2 hours were considered unsolved.

Table 3 sets forth the results of this experiment. Cells containing an 'x' represent unsolved problems. As shown in Table 3, the largest number of problems was solved by SCBR-cascading, and the fewest were solved by ALPINE. Unexpectedly, SPA-cbr-cascading performed no better than SPA-cbr. No method was able to solve problem 8.

Table 3. Problems solvable by each algorithm.

	Alpine	**SPA**	**SPA-cbr**	**SPA-cbr-cascading**	**SCBR**	**SCBR-cascading**
Prob1	√	√	√	√	√	√
Prob2	×	√	√	√	√	√
Prob3	√	√	√	√	√	√
Prob4	√	√	√	√	√	√
Prob5	√	√	√	√	√	√
Prob6	×	×	×	×	√	√
Prob7	×	×	√	√	√	√
Prob8	×	×	×	×	×	×
Prob9	√	√	√	√	√	√
Prob10	√	√	√	√	√	√
Prob11	√	√	√	√	√	√
Prob12	×	×	×	×	×	√
Prob13	×	×	×	×	×	√
Prob14	×	×	×	×	×	√

Solution Length The mean solution lengths of the six algorithms for the seven problems that all could solve is set forth in Figure 5. The variation among solution lengths was slight, with the shortest solutions produced by Alpine, SCBR, and SCBR-cascading.

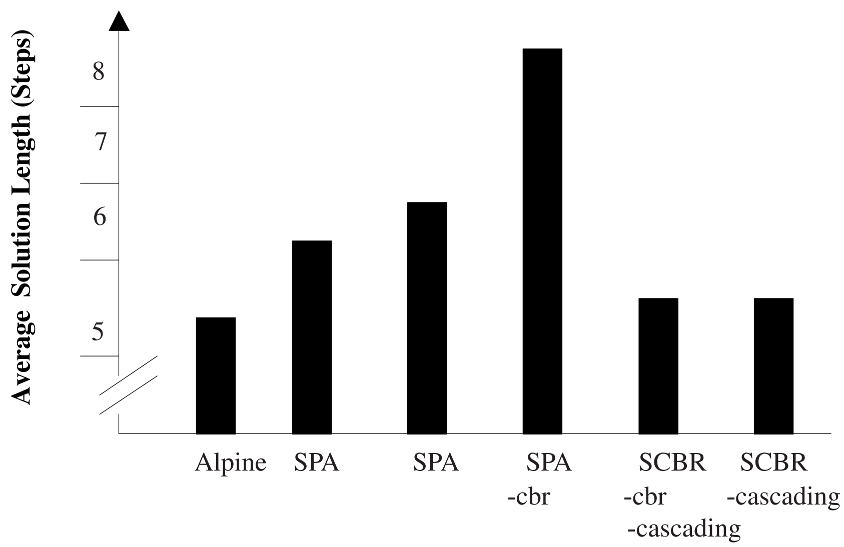


Fig. 5. Average Solution Length for the logistics problems

Execution Time The mean execution times of the six algorithms for the seven problems that all could solve is set forth in Figure 6. Alpine outperforms SPA, SPA-cbr, and SPA-cascading, but the lowest execution times were for SCBR and SCBR-cascading.

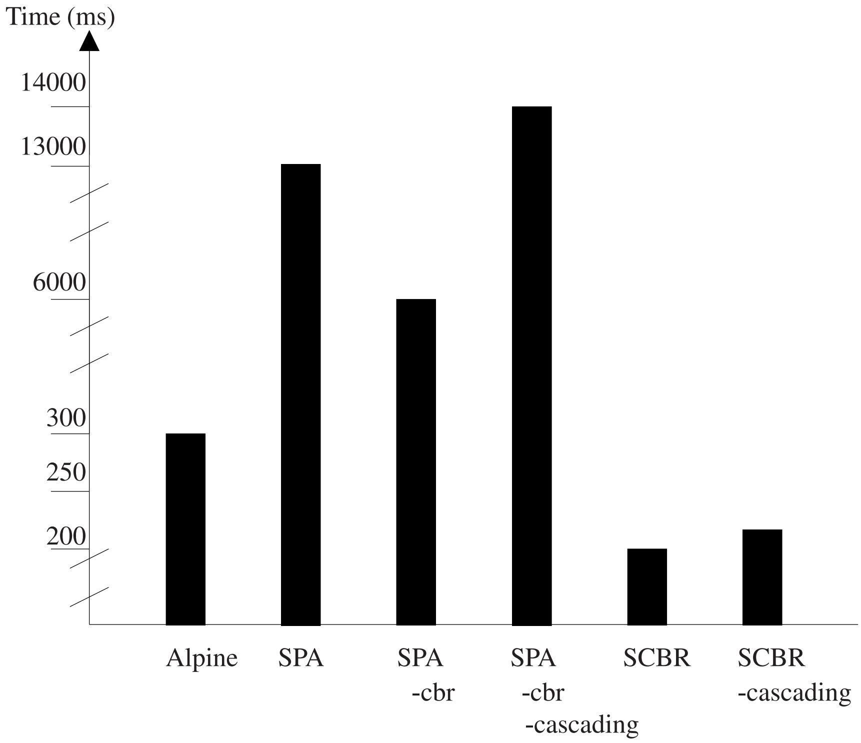


Fig. 6. Mean Execution Time for 7 Planning Problems

3.5 Summary

In the logistics domain, SCBR-cascading solved the largest number of cases—13 of the 14 cases—followed by SCBR, which solved 12 cases. SCBR and SCBR-cascading had the lowest execution times, and Alpine, SCBR, and SCBR-cascading tied for shortest solution lengths.

4 Discussion

The results of the evaluation in the logistic planning domain are preliminary because of the relatively small number of problems involved. However, the evaluation demonstrates that use of abstract cases for indexing can produce improvements the performance of planning systems given even a completely generic abstraction hierarchy and a general-purpose planner. The faster execution time of SCBR and SCBR-cascading than SPA-cbr and SPA-cbr-cascading given identical case libraries and an identical adaptation procedure indicates that SCBR led to retrieval of cases that were less expensive to adapt than the cases retrieved by SPA's ground-level retrieval mechanism.The shorter solution length for SCBR and SCBR-cascading, identical to the solution length for Alpine, also indicates that SCBR retrieved cases that more appropriate for each given problem.

The key distinction between the prefix-reversal task, a domain in which SCBR performed poorly, and the domains in which SCBR performed well, including the route-finding domain [4]and the logistics planning domain, appears to be that the optimal-case utility was high in the latter two problems but low in the prefix-reversal problem. The optimal-case utility was high in the logistics planning domain because the *fit-case* procedure was capable of adapting a case to a high-quality solution even to a very dissimilar problem. This is illustrated by the fact that SCBR-cascading did in fact solve every problem but one using *fit-case* as the adaptation mechanism. SCBR's ability to index the most adaptable case therefore led to greatly improved performance. Similarly, in the route-planning domain [4], the topography of the grids led to a high probability that any two cases would overlap and therefore to a high optimal-case utility. By contrast, cases were very unlikely to overlap in the prefix-reversal problem, and the adaptation mechanism was incapable of adapting arbitrary cases to solutions as good as could be obtained *ab initio.* As a result, the optimal-case utility, and therefore the performance of SCBR as measured by solution quality, was low.

It has been widely recognized that adaptability is a more important criterion for case retrieval than surface similarity to the current problem [17]. Unfortunately, the adaptation costs of a given case cannot, in general, be determined without actually performing the adaptation. However, the cost of adapting an abstraction of a case to an abstraction of the current problem can be a very accurate heuristic for ground-level adaptation costs. The logistics planning experiment demonstrates that SCBR is a general, domain-independent approach to retrieval by adaptation cost. The prefix-reveral experiment indicates, however, that SCBR nevertheless only reduces search cost when the adaptation mechanism and case-library size assures that optimal-case utility is high.

Acknowledgements

This research was supported by NSF Faculty Early Career Development Grant IRI-9502152, a German-American Fulbright Kommission Senior Scholar grant, a Flittie sabbatical grant, and by the University of Kaiserslautern Center for Learning Systems and Applications.

References

1. F. Bacchus and Q. Yang , *Downward Refinement and the Efficiency of Hierarchical Problem Solving*, Artificial Intelligence, 71:43-100, 1996.
2. R. Bergmann and W. Wilke, *On the Role of Abstraction in Case-Based Reasoning,* Advances in Case-Based Reasoning, Third European Workshop, Springer Verlag, 1996.

3. R. Bergmann and W. Wilke, *Building and Refining Abstract Planning Cases by Change of Representation Language,* Journal of Artificial Intelligence Research, 3:53-118, 1996.
4. L. K. Branting & D. W. Aha, *Stratified Case-Based Reasoning: Reusing Hierarchical Problem Solving Episodes,* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, August, pp. 20 – 25, 1995.
5. L. K. Branting*, Stratified Case-Based Reasoning in Non-Refinable Abstraction Hierarchies,* Proceedings of the Second International Conference on Case-Based Reasoning, Springer, pp. 519-530, July 1997.
6. A. Caprara, *Sorting by Reversals is difficult*, Proceedings of the First Annual International Conference on Computational Molecular Biology, pp. 75-83, January 1997.
7. J. G. Carbonell, C. A. Knoblock, and S. Minton. *Prodigy: An integrated architecture for planning and learning,* In K. Vanlehn, editor, Architectures for Intelligence, Erlbaum, Hillsdale, NJ, 1990.
8. J. Christensen. *A hierarchical planner that generates its own hierarchies.* Proceedings of the Eighth National Conference on Artificial Intelligence, pp.1004-1009, AAAI Press:Boston, MA (1990).
9. W. H. Gates, *Bounds for Sorting by Prefix Reversal,* Discrete Mathematics, 27:47-57 (1979).
10. S. Hanks and D. S. Weld, *A Domain-Independent Algorithm for Plan Adaptation,* Journal of Artificial Intelligence Research 2:319-360, 1995.
11. R. Holte, C. Drummond, M. Perez, R. Zimmer, and A. MacDonald, *Search With Abstractions: A Unifying Framework and New High-Performance Algorithm,* Proceedings of the Tenth Canadian Conference on Artificial Intelligence, Morgan Kaufmann, pp. 263-270 (1994).
12. S. Kambhampti and J. Hendler, *A Validation-Structure-Based Theory of Plan Modification,* Artificial Intelligence 55:193-258, 1992.
13. G.Klein and R. Calderwood, *How do People Use Analogues to Make Decisions?*, Proceedings of the DARPA Workshop on Case-Based Reasoning, Morgan Kaufman Publishers, Inc., 1988.
14. C. A. Knoblock, *Automatically Generating Abstractions for Planning*, Artificial Intelligence 68(2):243-302, 1994.
15. J. Setubal and J. Meidanis, Introduction to Computational Molecular Biology, PWS Publishing Co., 1997.
16. B. Smyth and P. Cunningham, *Deja Vu: A Hierarchical Case-Based Reasoning System for Software Design*, Proceedings of the European Conference on AI, John Wiley, pp. 587-589 (1992)
17. B. Smyth and M. Keane, *Adaptation-Guided Retrieval: Questioning the Similarity Assumption in Reasoning,* Artificial Intelligence (in press) 1999.
18. Y. Tao, *A Multiple-Domain Evaluation of Stratified Case-Based Reasoning*, M.S. Thesis, Department of Computer Science, University of Wyoming, August, 1998.